

**REQUERIMIENTO CORBA JAVA  
MANUAL DE INSTALACIÓN**



**Universidad  
del Cauca**

Vigilada Mineducación

**Integrantes**

**Luis Manuel Arango Guauña**

**Profesor**

**Pablo Augusto Magé Imbachí**

**Universidad del Cauca  
Facultad de Ingeniería Electrónica y Telecomunicaciones  
Programa de Ingeniería de Sistemas  
Curso: Sistemas Distribuidos  
Popayán, Marzo de 2022**

## Tabla de contenido

<b>I.</b>	<b>INTRODUCCIÓN.....</b>	<b>3</b>
<b>II.</b>	<b>Requerimientos del Sistema. ....</b>	<b>4</b>
<b>III.</b>	<b>Descripción y estructura de directorios.....</b>	<b>4</b>
<b>IV.</b>	<b>Descripción de los archivos fuente .....</b>	<b>4</b>
	Descripción “gusuarios.idl” .....	5
	a. Descripción personalDTO .....	6
	b. Descripción usuarioDTO .....	6
	c. Descripción credencialDTO .....	6
	d. Descripción AdmCllbckInt.....	6
	e. Descripción FapCllbckInt.....	6
	f. Descripción Interface GestionPersonalInt .....	7
	Descripción “gpacientes.idl” .....	7
	a. Descripción ValoracionDTO .....	8
	b. Descripción EjercicioDTO .....	8
	c. Descripción ProgramaDTO .....	8
	d. Descripción ProgramaFísicoDTO.....	8
	e. Descripción AsistenciaDTO.....	8
	f. Descripción notificaciónDTO .....	9
	g. Descripción Interface GestionPacientesInt .....	9
	Descripción s_gestion_usuarios.....	9
	a. Implementación GestionPersonalImpl .....	9
	b. Implementación del servidor de Objetos.....	14
	Descripción s_seguimiento_usuarios .....	15
	a. Implementación GestionPacientesImpl.java.....	15
	b. Implementación Servidor de Objetos.....	20
	Descripción cliente .....	21
	a. Implementación AdmCllbckImpl .....	21
	b. Implementación FapCllbckImpl .....	21
	c. Implementación Cliente de Objetos.....	21
	d. Implementación Utilidades de consola .....	22
	e. Implementación Métodos en el Cliente de Objetos .....	22
<b>V.</b>	<b>Descripción de compilación y ejecución .....</b>	<b>42</b>
<b>VI.</b>	<b>Repositorio GitHub .....</b>	<b>43</b>

## **I. INTRODUCCIÓN**

Este manual de instalación está diseñado para poder instalar correctamente un requerimiento de sistema distribuido utilizando el CORBA en el lenguaje Java para la asignatura de Sistemas distribuidos del Grupo A del programa de ingeniería de Sistemas de la Universidad del Cauca. Se presenta la elaboración que se utilizó para cubrir dicho requerimiento orientado al ejemplo del programa estilos de vida saludable especializándose en el programa de hora saludable enfocada para docentes y administrativos de las diferentes facultades de la Universidad del Cauca.

El programa hora saludable tendrá como clientes al administrador, quien será el encargado de registrar el personal y consultar dicho personal, otro de los clientes es la secretaria quien registra usuarios y puede consultarlos, el PAF es el usuario que puede hacer valoraciones sobre los pacientes registrados por la secretaria, finalmente los pacientes pueden ingresar para consultar las valoraciones, su programa físico y asistencias.

El programa está diseñado con un cliente de objetos y dos servidores de objetos, para la creación de ellos es necesario tener conocimientos básicos de compilación y mapeo de archivos IDL, para la implementación de este sistema usaremos 2 archivos IDL para la gestión del personal y la gestión de los pacientes.

## II. Requerimientos del Sistema.

Para la aplicación distribuida debe tener en cuenta los siguientes aspectos generales sobre las tecnologías utilizadas:

TECNOLOGÍA	NECESARIO
Sistema Operativo	Windows 10 / Linux
Lenguaje de Programación	JAVA
Java versión	Java 8
Java JDK versión	1.8.0_202
Modelo	CORBA de la OMG para GNU/Linux o Windows

Debemos disponer de un editor como lo es VS Code o un entorno para java de su preferencia, está desarrollado para mostrar resultados mediante la consola. Para esta guía presentaré un modelo a seguir utilizando Visual Studio Code.

## III. Descripción y estructura de directorios

Iniciemos por ubicar nuestro proyecto en una carpeta que será nuestro directorio principal, dele un nombre apropiado para la aplicación que se va a implementar, en este caso por regla de entrega de este requerimiento la he nombrado “lsdA\_corba\_archivos\_funte”, dentro de ella dispondremos de la siguiente organización como lo muestra la fig. 1



Fig. 1 Estructura de directorios

## IV. Descripción de los archivos fuente

A continuación, se describen las interfaces, clases y métodos utilizados para desarrollar el requerimiento. Empezaremos primero por definir los servidores que se van a ubicar en las carpetas “s\_gestion\_usuarios” y “s\_seguimiento\_usuarios” y “cliente”.

Primero en la carpeta raíz crearemos 2 nuevos archivos con la extensión “.idl”, estos nos permitirán en un paso próximo crear los archivos del soporte de CORBA.

### Descripción “gusuarios.idl”

IDL es una extensión es un lenguaje de descripción de interfaz, por lo que este archivo nos ayudará a definir las interfaces y estructuras que utilizaremos. Aquí tendremos la gestión del personal y el registro para los usuarios, así como también las estructuras necesarias para definir los objetos con los que trabajaremos. Así pues, crear primero un archivo llamado “gusuarios.idl” en la carpeta raíz(src), en seguida está definido nuestro IDL:

```
module sop_corba{
    interface AdmCllbckInt{
        void notificar(in string nombreCompleto, in long id );
    };
    interface FapCllbckInt{
        void informarIngreso(in string nombreCompleto, in long id);
    };
    struct personalDTO{
        string tipo_id;
        long id;
        string nombreCompleto;
        string ocupacion;
        string usuario;
        string clave;
    };
    struct usuarioDTO{
        long id;
        string nombreCompleto;
        string facultad;
        string tipo;
        string fechaIngreso;
        string patologia;
        string usuario;
        string clave;
    };
    struct credencialDTO{
        string usuario;
        string clave;
        long id;
    };
    interface GestionPersonalInt{
        void registrarPersonal(in personalDTO objPersonal, out boolean res);
        boolean consultarPersonal(in long id, out personalDTO objPersonal);
        void registrarUsuario(in usuarioDTO objUsuario, out boolean res);
        boolean consultarUsuario(in long id, out usuarioDTO objUsuario);
        boolean abrirSesion(in credencialDTO objCredencial);
        void registrarCallback(in AdmCllbckInt objCllbck);
        void registrarCallbackFap(in FapCllbckInt objCllbck);
    };
};
```

Como se puede observar no es un lenguaje de programación, sino que describe las interfaces y estructuras que hemos de utilizar para la construcción de este sistema distribuido.

Una vez creado el IDL procederemos a compilarlo y generar los soportes de CORBA para el servidor de gestión de usuario haciendo uso del siguiente comando:

```
idlj -fall -pkgPrefix sop_corba s_gestion_usuarios gusuaios.idl
```

El comando anterior genera los soportes de CORBA en la carpeta “sop\_corba” que está en “s\_gestion\_usuarios” utilizando el archivo IDL “gusuaios.idl”.

Ahora, describamos los struct e interfaces definidas en el archivo IDL anterior.

#### **a. Descripción personalDTO**

Este struct nos permite almacenar los datos referentes a un personal, dichos campos corresponden al tipo de identificación que puede tomar uno de los siguientes tres valores: CC, TI o PP, el número de la identificación que debe ser mayor a cero, el nombre completo del personal con una longitud mínima de 2 caracteres y máxima de 64, la ocupación del personal que en este caso solo pueden existir de 2 tipos: una secretaria (SEC) o un profesional de acondicionamiento físico (PAF), además solo puede existir un personal de cada ocupación, por último el personal tendrá asociado un nombre usuario y la clave, en ambos casos su longitud debe ser mínimo de 8 caracteres alfanuméricos.

#### **b. Descripción usuarioDTO**

Este struct nos permite almacenar los datos referentes a un usuario, dichos campos corresponden a el número de la identificación que debe ser mayor a cero, el nombre completo del personal con una longitud mínima de 2 caracteres y máxima de 64, la facultad a la que pertenece que puede tomar los siguientes valores: FACARTES, FACAGRO, FSALUD, FHUMANAS, FCCEA, FACNED, FDERECHO, FCIVIL, FIET; un campo que describe el tipo de cargo que tiene el usuario con dos posibles valores “Administrativo” o “Docente”, la fecha de ingreso en el que se registra, una patología que puede ser ingresada u omitida, por último el usuario tendrá asociado un nombre usuario y la clave, en ambos casos su longitud debe ser mínimo de 8 caracteres alfanuméricos.

#### **c. Descripción credencialDTO**

En credencialDTO almacenamos los datos correspondientes para poder acceder al sistema, esos datos son el usuario, contraseña y el ID asociado a los 2 anteriores.

#### **d. Descripción AdmCllbckInt**

Es interfaz es utilizada para hacer un callback al administrador cuando un personal PAF inicia sesión en el sistema, se notificara al administrador por medio del método notificar().

#### **e. Descripción FapCllbckInt**

Es utilizada para hacer un callback y notificar el ingreso de un nuevo paciente al PAF cuando una secretaria registra a un nuevo paciente.

#### **f. Descripción Interface GestionPersonalInt**

Describe los métodos usados para gestionar el registro y consulta de un personal, el registro y consulta de un usuario, el acceso (login) al sistema, y el registro de los callback para el administrador y el FAP.

#### **Descripción “gpacientes.idl”**

Aquí tendremos la gestión de los pacientes, así como también las estructuras necesarias para definir los objetos con los que trabajaremos. Así pues, crear primero un archivo llamado “gpacientes.idl” en la carpeta raíz(src), en seguida está definido nuestro IDL:

```
module sop_corba{
    struct AsistenciaDTO{
        long idPaciente;
        string fechaAsistencia;
        string observacion;
    };
    typedef sequence<AsistenciaDTO> ArrayAsistencia;
    struct EjercicioDTO{
        string nombreEjercicio;
        long repeticiones;
        long peso;
    };
    typedef sequence<EjercicioDTO> ArrayEjercicios;
    struct ProgramaDTO{
        long dia;
        ArrayEjercicios listaEjercicios;
    };
    typedef sequence<ProgramaDTO> ArrayPrograma;
    struct ProgramaFisicoDTO{
        long idPaciente;
        string fechaInicio;
        ArrayPrograma listaProgramaSemana;
    };
    struct ValoracionDTO{
        long idPaciente;
        string fechaValoracion;
        long fecCardiacaReposo;
        long fecCardiacaActiva;
        long estatura;
        long brazo;
        long pierna;
        long pecho;
        long cintura;
        string estado;
    };
    struct notificacionDTO{
        string nombreCompleto;
        string ocupacion;
    };
    interface GestionPacientesInt{
        void registrarValoracion(in ValoracionDTO objValoracion, out boolean res);
        boolean consultarValoracion(in long id, out ValoracionDTO objValoracion);
    };
}
```

```

    void registrarProgramaFisico(in ProgramaFisicoDTO objProgramaFisico, out
boolean res);
    boolean consultarProgramaFisico(in long id, out ProgramaFisicoDTO
objProgramaFisico);
    void registrarAsistencia(in AsistenciaDTO objAsistencia, out boolean res);
    boolean consultarAsistencia(in long id, out ArrayAsistencia lstAsistencia);
    void contarFaltas(in long id, out long numFaltas);
    void enviarNotificacion(in notificacionDTO objNotificacion);
};
};

```

Una vez creado el IDL procederemos a compilarlo y generar los soportes de CORBA para el servidor de gestión de pacientes haciendo uso del siguiente comando:

```
idlj -fall -pkgPrefix sop_corba s_seguimiento_usuarios gpacientes.idl
```

El comando anterior genera los soportes de CORBA en la carpeta “sop\_corba” que está en “s\_seguimiento\_usuarios” utilizando el archivo IDL “gusuaio.idl”.

Explicamos a continuación los struct del IDL.

#### **a. Descripción ValoracionDTO**

Ester struct nos permite almacenar la información de una valoración realizada por el PAF a un paciente, contiene el id del paciente, la fecha de la valoración, la frecuencia cardiaca en reposo (pulsaciones por minuto), frecuencia cardiaca activa, las mediciones en centímetros de: estatura, brazo, pierna, pecho y cintura, finalmente un campo en el que se almacena con base en los campos anteriores el estado físico del paciente.

#### **b. Descripción EjercicioDTO**

Nos permite almacenar la información de un ejercicio que se compone del nombre del mismo, la cantidad de repeticiones a realizar y el peso en kilogramos con el que se ejecuta el ejercicio.

#### **c. Descripción ProgramaDTO**

Contiene la información del programa de ejercicios para un día en específico, los atributos que lo describen son el número del día y una lista de ejercicios que se realizan para ese día.

#### **d. Descripción ProgramaFísicoDTO**

Describe el programa físico debe desarrollar un paciente, y se identifica por el ID del paciente, la fecha de inicio del programa físico y una lista del tipo Programa DTO que contiene los días y los ejercicios a realizar.

#### **e. Descripción AsistenciaDTO**

Almacena la información de la asistencia al programa físico para un paciente, por el id del paciente se identifica y se registra la asistencia, la fecha en la que se registra la asistencia y una observación que puede ser una excusa, una no asistencia, o ninguna en caso de asistir.



#### f. Descripción notificaciónDTO

Contiene el nombre y la ocupación de un personal con permiso para ingresar al servidor de seguimiento de usuarios.

#### g. Descripción Interface GestionPacientesInt

Describe los recursos que van a consumir el PAF y el paciente, dichos recursos los definen los métodos para registrar y consultar una valoración, registrar y consultar un programa físico, registrar y consultar asistencias, un método para hacer el conteo del número de faltas de un paciente por su ID y por último un método para notificar el acceso del PAF.

#### Descripción s\_gestion\_usuarios

Pasamos ahora a implementar los soportes que hemos generado con los IDL, ubicados en la carpeta *src/s\_gestion\_usuarios/servidor* creamos un nuevo archivo java que nombraremos como “*GestionPersonalImpl*”, y otro más “*ServidorDeObjetos*”.

#### a. Implementación GestionPersonalImpl

Lo primero que hacemos es extender esta clase de *GestionPersonalIntPOA* (Esta clase se habrá generado al momento de compilar con el comando del IDL *gusuarios.idl*)

##### 1. Atributos

Los Atributos que debemos definir en esta clase serán los siguientes:

```
public class GestionPersonalImpl extends GestionPersonalIntPOA{
    //Lista del personal registrado
    private ArrayList<personalDTO> lstPersonal;
    //Lista de los usuarios(pacientes) registrados
    private ArrayList<usuarioDTO> lstUsuarios;

    GestionPacientesInt ref;
    //Referencia al callback Admin
    private AdmCllbckInt admCllbckInt;
    //Referencia al callback Admin
    private FapCllbckInt fapCllbckInt;
    //Atributos del administrador
    String admNombre = "Administrador";
    String admTipoID = "CC";
    int admID = 0;
    String admOcup = "Admin";
    String admUser = "Admin";
    String admPsw = "12345";
    personalDTO admin = new personalDTO(admTipoID,admID,admNombre, admOcup, admUser, admPsw);
}
```

Fig. 2 – Atributos *GestionPersonalImpl*

##### 2. Constructor

Inicializamos el constructor con los parámetros definidos en el paso anterior.

```

public GestionPersonalImpl(){
    super();
    lstPersonal = new ArrayList<personalDTO>();
    lstUsuarios = new ArrayList<usuarioDTO>();
    lstPersonal.add(admin);
    admCllbckInt = null;
    fapCllbckInt = null;
    ref = null;
}

```

Fig. 3 – Constructor de la clase *GestionPersonalImpl*

### 3. Método auxiliar *existeID*

Este método está hecho para verificar si ya existe un id registrado en el sistema, esto será importante para poder tener integridad en los datos y que no existan ID repetidos en los registros del personal y el usuario. True si ya existe o false de lo contrario.

```

private boolean existeID(int id){
    boolean resultado = false;
    for (int i = 0; i < lstPersonal.size(); i++) {
        if (lstPersonal.get(i).id == id) {
            resultado = true;
        }
    }

    for (int j = 0; j < lstUsuarios.size(); j++) {
        if (lstUsuarios.get(j).id == id) {
            resultado = true;
        }
    }
    return resultado;
}

```

Fig. 4 – Método auxiliar *existeID()*

### 4. Método auxiliar *existeRegistroPersonal*

Este método está hecho para verificar si ya existe un personal con una ocupación específica, ya que se tiene la restricción de que solo pueden existir una secretaria y un PAF. True si ya existe un personal con esa ocupación y false de lo contrario

```

private boolean existeRegistroPersonal(personalDTO prmObjPersonaLDTO){
    boolean resultado = false;
    for (int i = 0; i < lstPersonal.size(); i++) {
        if (lstPersonal.get(i).ocupacion.equals(prmObjPersonaLDTO.ocupacion)) {
            resultado = true;
        }
    }
    return resultado;
}

```

Fig. 5 – Método auxiliar *existeRegistroPersonal()*

### 5. Método auxiliar existeRegistroPersonal

Este método nos será útil en un paso posterior, nos permitirá hacer la conexión con el segundo servidor de seguimiento a usuarios al obtener la referencia remota.

```
public void consultarReferenciaRemota(NamingContextExt nce, String servicio){
    //GestionNotificaciones ref;
    try{
        this.ref = GestionPacientesIntHelper.narrow(nce.resolve_str(servicio));
        System.out.println("Obtenido el manejador sobre el servidor de objetos: ");
    }catch(Exception ex){
        System.out.println("Error: "+ ex.getMessage());
    }
}
```

Fig. 6 – Método auxiliar consultarReferenciaRemota()

### 6. Método Override registrarCallbackFap

Nos permite asignar la referencia para el FAP que inicializamos en nulo en el constructor de esta clase.

```
@Override
public void registrarCallbackFap(FapCllbckInt objCllbck) {
    System.out.println("*** Desde registrarCallbackFap() ***");
    this.fapCllbckInt = objCllbck;
}
```

Fig. 7 – registrarCallbackFap()

### 7. Método Override registrarCallback

Este método nos permite asignar la referencia para el Administrador que inicialmente esta en nulo en el constructor de esta clase.

```
@Override
public void registrarCallback(AdmCllbckInt objCllbck) {
    System.out.println("*** Desde registrarCallback() ***");
    this.admCllbckInt = objCllbck;
}
```

Fig. 8 – RegistrarCallbackFap()

### 8. Método Override registrarPersonal

Nos permite registrar un objeto de tipo personalDTO, primero verifica si hay espacio en la lista del personal seguido a ello verifica si el personal con la ocupación entrante ya se encuentra previamente registrado, y finalmente si ese personal con el ID no existe en los registros del personal y usuarios se procede a hacer la adición de ese objPersonal a la lista

del personal. En el parámetro de salida *res* se almacena el resultado de la operación, true de ser exitosa y false de lo contrario

```
@Override
public void registrarPersonal(personalDTO objPersonal, BooleanHolder res) {
    System.out.println("*** En registrarPersonal()...");
    res.value = false;
    if (lstPersonal.size() < 3) {
        if(!existeRegistroPersonal(objPersonal)){
            if (!existeID(objPersonal.id)) {
                if(lstPersonal.add(objPersonal)){
                    res.value = true;
                    System.out.println("El personal " + objPersonal.nombreCompleto + " y ocupacion "
                }
            }else{
                System.out.println("No se puede registrar un usuario con un id existente en el sistema");
            }
        }else{
            System.out.println("ya existe un personal con la ocupacion " + objPersonal.ocupacion);
        }
    }else{
        System.out.println("Personal No registrado, se alcanzo la cantidad maxima de personas a registrar");
    }
}
```

Fig. 9 – registrarPersonal()

#### 9. Método Override registrarUsuario

Nos permite registrar un objeto de tipo usuarioDTO, primero verifica si ya existe algún usuario con el id proporcionado y si la inserción es exitosa el usuario podrá ser registrado en el sistema. En el parámetro de salida *res* se almacena el resultado de la operación, true de ser exitosa y false de lo contrario

```
@Override
public void registrarUsuario(usuarioDTO objUsuario, BooleanHolder res) {
    System.out.println("*** En registrarUsuario()...");
    res.value = false;
    if (!existeID(objUsuario.id)) {
        if(lstUsuarios.add(objUsuario)){
            res.value = true;
            System.out.println("El Usuario " + objUsuario.nombreCompleto + " fue registrado con exito");
            fapCllbckInt.informarIngreso(objUsuario.nombreCompleto, objUsuario.id);
        }
    }else{
        System.out.println("No se puede registrar un usuario con un id existente en el sistema");
    }
}
```

Fig. 10 – registrarUsuario()

#### 10. Método Override consultarPersonal

Nos permite consultar si un personal está registrado en el sistema, la consulta se realiza partiendo del ID, si hay coincidencia se retorna el objeto de tipo personal y true, de lo contrario si no existe se retornará un objeto de tipo personal con parámetros por defecto y false.

```

@Override
public boolean consultarPersonal(int id, personalDTOHolder objPersonal) {
    System.out.println("*** En consultarPersonal()...");
    personalDTO varObjPersonal = new personalDTO("", 0, "", "", "", "");
    objPersonal.value = varObjPersonal;
    boolean resultado = false;
    try {
        if (!lstPersonal.isEmpty()) {
            for (int i = 0; i < lstPersonal.size(); i++) {
                if (lstPersonal.get(i).id == id) {
                    objPersonal.value = lstPersonal.get(i);
                    resultado = true;
                    System.out.println("*** Se encontro al usuario con id " + lstPersonal.get(i).id);
                    break;
                }
            }
        } else {
            System.out.println("No hay personal registrado en el sistema");
        }
    } catch (Exception e) {
        System.out.println("NO se pudo recuperar la consulta");
    }
    return resultado;
}

```

Fig. 11 – consultarPersonal()

### 11. Método Override consultarUsuario

Nos permite consultar si un usuario está registrado en el sistema, la consulta se realiza partiendo del ID, si hay coincidencia se retorna el objeto de tipo usuario y true, de lo contrario si no existe se retornará un objeto de tipo usuario con parámetros por defecto y false.

```

@Override
public boolean consultarUsuario(int id, usuarioDTOHolder objUsuario) {
    System.out.println("*** En consultarUsuario()...");
    usuarioDTO vUsuarioDTO = new usuarioDTO(0, "", "", "", "", "", "", "");
    objUsuario.value = vUsuarioDTO;
    boolean resultado = false;
    try {
        if (!lstUsuarios.isEmpty()) {
            for (int i = 0; i < lstUsuarios.size(); i++) {
                if (lstUsuarios.get(i).id == id) {
                    objUsuario.value = lstUsuarios.get(i);
                    resultado = true;
                    System.out.println("*** Se encontro al usuario con id " + id);
                    break;
                }
            }
        } else {
            System.out.println("No hay usuarios registrados en el sistema");
        }
    } catch (Exception e) {
        System.out.println("NO se pudo recuperar la consulta");
    }
    return resultado;
}

```

Fig. 12 – ConsultarUsuario()

## 12. Método Override *abrirSesion*

En este método retornamos una respuesta true o false respecto a una petición de acceso al sistema, mediante un objeto de tipo credencial verificamos que el usuario que hace la petición esté autorizado para ingresar al sistema. True en caso de que la operación sea exitosa y false de lo contrario. En el caso del FAP se envía una notificación al Admin y se construye un objeto *notificacionDTO* para informar al servidor de su ingreso al sistema mediante a referencia remota *ref*.

```
@Override
public boolean abrirSesion(CredencialDTO objCredencial) {
    System.out.println("***En abrirSesion()...");
    boolean resultado = false;
    personalInfo varPersonal = null;
    for (int i = 0; i < lstPersonal.size(); i++) {
        if (objCredencial.usuario.equals(lstPersonal.get(i).usuario) &&
            objCredencial.clave.equals(lstPersonal.get(i).clave) &&
            objCredencial.id == lstPersonal.get(i).id) {
            resultado = true;
            varPersonal = lstPersonal.get(i);
            break;
        }
    }

    usuarioDTO varUsuario = null;
    for (int j = 0; j < lstUsuarios.size(); j++) {
        if (objCredencial.usuario.equals(lstUsuarios.get(j).usuario) &&
            objCredencial.clave.equals(lstUsuarios.get(j).clave) &&
            objCredencial.id == lstUsuarios.get(j).id) {
            resultado = true;
            varUsuario = lstUsuarios.get(j);
            break;
        }
    }

    if (varUsuario != null) {
        System.out.println("El usuario " + varUsuario.usuario + " ingresó al Sistema");
    }
    if (varPersonal != null) {
        String nombreCompleto = varPersonal.nombreCompleto;
        String ocupacion = varPersonal.ocupacion;

        switch (ocupacion) {
            case "Admin":
                System.out.println("Admin ingresó al Sistema");
                break;
            case "SEC":
                System.out.println("La secretaria " + nombreCompleto + " ingresó al Sistema");
                break;
            case "PAC":
                this.admCllckInt.notificar(nombreCompleto, varPersonal.id);
                System.out.println("El #de " + nombreCompleto + " inició Sesión");
                System.out.println("Enviando notificación()...");
                notificacionDTO varNotificacion = new notificacionDTO();
                varNotificacion.nombreCompleto = nombreCompleto;
                varNotificacion.ocupacion = ocupacion;
                ref.EnviarNotificacion(varNotificacion);
                break;
        }
    }

    return resultado;
}
```

Fig. 13 – *abrirSesion()*

### b. Implementación del servidor de Objetos

Ubicados en la siguiente ruta de la carpeta raíz de nuestro directorio *src/s\_gestion\_usuario/servidor* creamos una clase denominada *ServidorDeObjetos*, esta clase será el punto de arranque del servidor actual. Definamos a continuación esta clase:

```
public class ServidorDeObjetos {
    RunOnDmg
    public static void main(String args[]){
        try {
            ORB orb = ORB.init(args, null);
            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();

            GestionPersonalImpl personalImpl = new GestionPersonalImpl();
            //personalImpl.setORB(orb);

            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(personalImpl);
            GestionPersonalInt cref = GestionPersonalIntHelper.narrow(ref);

            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            String name = "objRemotoPersonal";
            NameComponent path[] = ncRef.to_name(name);
            ncRef.rebind(path, cref);

            System.out.println("consultando Referencia Remota...");
            personalImpl.consultarReferenciaRemota(ncRef, "objRemotoPacientes");
            System.out.println("El servidor gestion de usuarios Inicio y esta esperando Acciones");
            orb.run();
        } catch (Exception e) {
            System.err.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

Fig. 14 – *ServidorDeObjetos()*

Como pueden observar, esta clase instancia el orb y es registrado con el nombre “*objRemotoPersonal*”, y luego de ello consulta la referencia remota del servidor 2 que estaremos describiendo en la sección posterior, con esto el servidor queda en escucha. Aclarar que primero se debe lanzar el ORBD y el servidor 2, esta secuencia se describirá posteriormente.

### Descripción s\_seguimiento\_usuarios

Pasamos ahora a implementar los soportes que hemos generado con el IDL “*gpacientes.idl*”, ubicados en la carpeta *src/s\_seguimiento\_usuarios/servidor* creamos un nuevo archivo java que nombraremos como “*GestionPacientesImpl*”, y otro más “*ServidorDeObjetos*”.

#### a. Implementación GestionPacientesImpl.java

Lo primero que hacemos es extender esta clase de *GestionPacientesIntPOA* (Esta clase se habrá generado al momento de compilar con el comando del IDL *gpacientes.idl*).

##### 1. Atributos y constructor

Los Atributos que debemos definir en esta clase serán los siguientes y en el constructor inicializamos dichos atributos:

```
public class GestionPacientesImpl extends GestionPacientesIntPOA{

    private ArrayList<ValoracionDTO> lstValoraciones;
    private ArrayList<ProgramaFisicoDTO> lstProgramas;
    private ArrayList<AsistenciaDTO> lstAsistencia;

    public GestionPacientesImpl() {
        super();
        this.lstValoraciones = new ArrayList<ValoracionDTO>();
        this.lstProgramas = new ArrayList<ProgramaFisicoDTO>();
        this.lstAsistencia = new ArrayList<AsistenciaDTO>();
    }
}
```

Fig. 15 - *GestionPacientesImpl*

##### 2. Método auxiliar buscarValoracion

Este método busca mediante un id de paciente una valoración asociada a ese ID, si la encuentra retornara la posición, es decir, el índice de la lista de valoraciones donde se encuentra ubicada esa valoración, y devolverá -1 en caso de que la búsqueda no arroje coincidencias.

```
public int buscarValoracion(int id){
    int resultado = -1;
    for (int i = 0; i < lstValoraciones.size(); i++) {
        if (this.lstValoraciones.get(i).idPaciente == id) {
            resultado = i;
            break;
        }
    }
    return resultado;
}
```

Fig. 16 – *buscarValoracion()*

### 3. Método auxiliar buscarPrograma

Este método busca en la lista de programas un objeto asociado al id de un paciente, de haber coincidencia se retorna el índice de la lista donde se encuentra ubicado, si la operación no encuentra ningún resultado retornará el valor -1.

```
public int buscarPrograma(int id){
    int resultado = -1;
    for (int i = 0; i < lstProgramas.size(); i++) {
        if (this.lstProgramas.get(i).idPaciente == id) {
            resultado = i;
            break;
        }
    }
    return resultado;
}
```

Fig. 17 – bucarPrograma()

### 4. Método auxiliar eliminarValoracion

Este método busca en la lista de valoraciones un objeto asociado al id de un paciente, de haber coincidencia se procede a eliminar el objeto de la lista, retornará true si la eliminación tuvo éxito o false de lo contrario.

```
public boolean eliminarValoracion(int id){
    boolean resultado = false;
    for (int i = 0; i < this.lstValoraciones.size(); i++) {
        if(this.lstValoraciones.get(i).idPaciente == id){
            this.lstValoraciones.remove(i);
            resultado = true;
            break;
        }
    }
    return resultado;
}
```

Fig. 18 – eliminarValoracion()

### 5. Método auxiliar eliminarPrograma

Este método busca en la lista de programas un objeto asociado al id de un paciente, de haber coincidencia se procede a eliminar el objeto de la lista, retornará true si la eliminación tuvo éxito o false de lo contrario.

```
public boolean eliminarPrograma(int id){
    boolean resultado = false;
    for (int i = 0; i < this.lstProgramas.size(); i++) {
        if (this.lstProgramas.get(i).idPaciente == id) {
            this.lstProgramas.remove(i);
            resultado = true;
            break;
        }
    }
    return resultado;
}
```

Fig. 19 – eliminarPrograma()



#### 6. Método auxiliar generarEstado

Este método define el estado de una persona, dependiendo de las pulsaciones por minuto de la frecuencia cardiaca en estado de reposo se puede considerar el estado de salud de un paciente, los posibles valores que puede retornar son: Enfermo, Regular o Sano.

```
public String generarEstado(ValoracionDTO objValoracion){
    String estado = "";
    int FCR = objValoracion.fecCardiacaReposo;
    if (FCR >= 86 || FCR <= 50) {
        estado = "Enfermo";
    }
    if (FCR >= 75 && FCR < 86) {
        estado = "Regular";
    }
    if (FCR > 50 && FCR < 75) {
        estado = "Sano";
    }
    return estado;
}
```

Fig. 20 – generarEstado()

#### 7. Método Override registrarValoracion

Este método registra una valoración en la lista de valoraciones, de ser exitoso el proceso de registro retornará en su parámetro de salida *res* true, false de lo contrario.

```
@Override
public void registrarValoracion(ValoracionDTO objValoracion, BooleanHolder res) {
    System.out.println("***Entrando a registrarValoracion()...");
    res.value = false;
    if(objValoracion != null){
        objValoracion.estado = generarEstado(objValoracion);
        res.value = lstValoraciones.add(objValoracion);
        if (res.value) {
            System.out.println("Valoracion fisica Registrada! ");
        }else{
            System.out.println("ERROR: No se pudo registrar la valoracion fisica.");
        }
    }
}
```

Fig. 21 – registrarValoracion()

#### 8. Método Override consultarValoracion

Este método consulta una valoración por el id de paciente asociado a ella, retornará True y el objeto de tipo valoración en el parámetro *objValoracion* si se encontró una coincidencia, y si falla retornará false y el *objValoracion* tendrá valores por defecto.

```

@Override
public boolean consultarValoracion(int id, ValoracionDTOHolder objValoracion) {
    System.out.println("***Entrando a consultarValoracion()...");
    ValoracionDTO varObjValoracionDTO = new ValoracionDTO(0, "", 0, 0, 0, 0, 0, 0, 0, "");
    objValoracion.value = varObjValoracionDTO;
    boolean resultado = false;
    int bandera = buscarValoracion(id);
    if (bandera != -1) {
        objValoracion.value = lstValoraciones.get(bandera);
        resultado = true;
    } else {
        System.out.println("ERROR: No se pudo encontrar la valoracion.");
    }
    return resultado;
}

```

Fig. 22 – consultarValoracion()

### 9. Método Override registrarPrograma

Este método registra una valoración en la lista de valoraciones, de ser exitoso el proceso de registro retornará en su parámetro true, y false de lo contrario.

```

@Override
public void registrarProgramaFisico(ProgramaFisicoDTO objProgramaFisico, BooleanHolder res) {
    System.out.println("***Entrando a regProgramaFisico()...");
    res.value = false;
    if (objProgramaFisico != null) {
        res.value = lstProgramas.add(objProgramaFisico);
        if (res.value) {
            System.out.println("Programa fisico Agregado! ");
        } else {
            System.out.println("ERROR: No se pudo registrar el programa fisico.");
        }
    }
}

```

Fig. 23 – registrarProgramaFisico

### 10. Método Override consultarProgramaFisico

Este método consulta un programa por el id de paciente asociado a ella, retornará True y el objeto de tipo ProgramaFisico en el parámetro objProgramaFisico si se encontró una coincidencia, y si falla retornará false y el objProgramaFisico tendrá valores por defecto.

```

@Override
public boolean consultarProgramaFisico(int id, ProgramaFisicoDTOHolder objProgramaFisico) {
    System.out.println("***Entrando a consultarProgramaFisico()...");
    ProgramaFisicoDTO[] varListaProgramasDefault = {};
    ProgramaFisicoDTO varObjProgramaFisicoDTO = new ProgramaFisicoDTO(0, "", varListaProgramasDefault);
    objProgramaFisico.value = varObjProgramaFisicoDTO;
    boolean resultado = false;
    int bandera = buscarPrograma(id);
    if (bandera != -1) {
        objProgramaFisico.value = lstProgramas.get(bandera);
        resultado = true;
    } else {
        System.out.println("ERROR: No se pudo encontrar el Programa.");
    }
    return resultado;
}

```

Fig. 24 – ConsultarProgramaFisico()

### 11. Método Override registrarAsistencia

Este método registra una asistencia en la lista de Asistencias, de ser exitoso el proceso de registro retornará en su parámetro *res* true, y false de lo contrario. Hace uso del conteo de faltas, y cuando las son iguales a 3 hace uso de los métodos auxiliares para borrar la Valoración y el programa físico del Paciente con el ID asociado a la asistencia.

```
@Override
public void registrarAsistencia(AsistenciaDTO objAsistencia, BooleanHolder res) {
    System.out.println("***En registrarAsistencia()...");
    res.value = false;
    if (objAsistencia != null) {
        res.value = this.lstAsistencia.add(objAsistencia);
        IntHolder faltas = new IntHolder();
        contarFaltas(objAsistencia.idPaciente, faltas);
        if (res.value) {
            System.out.println("Registro exitoso de asistencia.");
        } else {
            System.out.println("Registro de asistencia fallo.");
        }
        if (faltas.value >= 3) {
            System.out.println("Alcanzo el numero maximo de faltas");
            System.out.println("Se borrarán valoración y programa físico");
            eliminarValoracion(objAsistencia.idPaciente);
            eliminarPrograma(objAsistencia.idPaciente);
        }
    }
}
```

Fig. 25 – registrarAsistencia

### 12. Método Override consultarAsistencia

Este método consulta las asistencias de un paciente por su ID, si el paciente tiene registros se irán almacenando en el parámetro *lstAsistencia* y retornará true, en caso contrario *lstAsistencia* tendrá un objeto de tipo lista de asistencias con parámetros por defecto y el retorno será false.

```
@Override
public boolean consultarAsistencia(int id, ArrayAsistenciaHolder lstAsistencia) {
    System.out.println("***En consultarAsistencia()...");
    AsistenciaDTO varAsistenciaDTO = new AsistenciaDTO(0, "", "");
    AsistenciaDTO[] lstAsistenciaDefault = {varAsistenciaDTO};
    lstAsistencia.value = lstAsistenciaDefault;
    boolean resultado = false;
    if (this.lstAsistencia.isEmpty()) {
        System.out.println("La lista de valoraciones esta vacia");
    } else {
        ArrayList<AsistenciaDTO> lstAsistenciasTemp = new ArrayList<AsistenciaDTO>();
        for (int i = 0; i < this.lstAsistencia.size(); i++) {
            if (this.lstAsistencia.get(i).idPaciente == id) {
                lstAsistenciasTemp.add(this.lstAsistencia.get(i));
            }
        }
        if (lstAsistenciasTemp.size() > 0) {
            lstAsistenciaDefault = new AsistenciaDTO[lstAsistenciasTemp.size()];
            lstAsistencia.value = lstAsistenciaDefault;
            for (int j = 0; j < lstAsistenciasTemp.size(); j++) {
                lstAsistencia.value[j] = lstAsistenciasTemp.get(j);
            }
            resultado = true;
        }
    }
    return resultado;
}
```

Fig. 26 – consultarAsistencia ()

### 13. Método Override contarFaltas

Este método cuenta en las faltas de un paciente por su ID, lo realiza haciendo una búsqueda cíclica en la lista de asistencia, retorna el número de faltas para el paciente y 0 cuando no se encuentran faltas reportadas

```
@Override
public void contarFaltas(int id, IntHolder numFaltas) {
    System.out.println("***En contarFaltas()...");
    numFaltas.value = 0;
    for (int i = 0; i < this.lstAsistencia.size(); i++) {
        if (this.lstAsistencia.get(i).idPaciente == id) {
            if (this.lstAsistencia.get(i).observacion.equals("No asiste")){
                numFaltas.value = numFaltas.value + 1;
            }
        }
    }
}
```

Fig. 27 – contarFaltas()

### 14. Método Override enviarNotificacion

Envía una notificación generando un eco al servidor de seguimiento de usuarios, este se utiliza cuando el FAP inicia sesión, notificando así que ha ingresado al sistema que está habilitado para él

```
@Override
public void enviarNotificacion(notificacionDTO objNotificacion){
    System.out.println("***En enviarNotificacion()...");
    System.out.println("El personal [" + objNotificacion.nombreCompleto + "] y ocupacion [" +
        objNotificacion.ocupacion + "] esta autorizado para ingresar al sistema.");
}
```

Fig. 28 – enviarNotificacion()

## b. Implementación Servidor de Objetos

```
public class ServidorDeObjetos {
    Run()Debug;
    public static void main(String args[]){
        try {
            ORB orb = ORB.init(args, null);

            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();

            GestionPacientesImpl pacientesImpl = new GestionPacientesImpl();
            //pacientesImpl.setORB(orb);

            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(pacientesImpl);
            GestionPacientesInt cref = GestionPacientesIntHelper.narrow(ref);

            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            String name = "objRemotoPacientes";
            NameComponent path[] = ncRef.to_name(name);
            ncRef.rebind(path, cref);

            System.out.println("El servidor gestion de usuarios Inicio y esta esperando Acciones");
            orb.run();
        }
        catch (Exception e) {
            System.err.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

## Descripción cliente

Pasamos ahora a implementar los soportes que hemos generado con los IDL, ubicados en la carpeta *src/cliente/cliente* creamos un nuevo archivo java que nombraremos como “*AdmCllbckImpl*” y “*FapCllbckImpl*”, y en *src/cliente* creamos un archivo nombrado “*ClienteDeObjetos*”.

### a. Implementación AdmCllbckImpl

Extendemos de *AdmCllbckIntPOA* (Archivo generado en el soporte de CORBA) y procedemos a implementar el método notificar, que es una alerta que imprime en consola para el administrador cuando un FAP ingresa

```
public class AdmCllbckImpl extends AdmCllbckIntPOA{
    public AdmCllbckImpl(){
        super();
    }
    @Override
    public void notificar(String nombreCompleto, int id) {
        System.out.println("*** objeto callback desde notificar()...");
        System.out.println("***El usuario [" + nombreCompleto + "] identificado con id [" +
            id + "] ingreso a la aplicacion***");
        System.out.println("== == == == ==");
    }
}
```

Fig. 29 – Class AdmCllbckImpl

### b. Implementación FapCllbckImpl

Extendemos de *FapCllbckIntPOA* (Archivo generado en el soporte de CORBA) y procedemos a implementar el método Informar ingreso, que se hace cuando un nuevo paciente es registrado, se le notifica así pues al PAF para que pueda hacer la respectiva valoración.

```
public class FapCllbckImpl extends FapCllbckIntPOA{
    public FapCllbckImpl(){
        super();
    }
    @Override
    public void informarIngreso(String nombreCompleto, int id) {
        System.out.println("*** objeto callback desde informarIngreso()...");
        System.out.println("***El usuario [" + nombreCompleto + "] identificado con id [" +
            id + "] esta disponible para la valoracion***");
        System.out.println("== == == == ==");
    }
}
```

Fig. 30 – Class FapCllbckImpl

### c. Implementación Cliente de Objetos

En esta clase tendremos primero el arranque del cliente además de los menús y el paso de información a los servidores. En pocas palabras esta clase consume todos los servicios de los servidores dependiendo del tipo de usuario, aquí se arman los paquetes de datos de informan a pasar en cada uno de los métodos. También se controlan las restricciones como longitudes

de las cadenas y valores nulos o negativos, para que la información pase filtrada a los servidores.

En primera medida declaramos las referencias de los servidores y obtenemos las referencias remotas mediante el nombre que hemos configurado en los servidores de objetos anteriormente, luego obtenemos la referencia para los callback y hacemos llamado al método de menú principal.



```

public class ClienteDeObjetos {
    public static GestionPersonalInt href1;
    public static GestionPacientesInt href2;
    RunData;
    public static void main(String args[]) {
        try {
            // Crear e inicializar el orb
            ORB orb = ORB.init(args, null);
            // obtenemos el nombre de servidor vali de name service
            org.omg.CORBA.Object ref = orb.resolve_initial_references("NameService");
            // Use NamingContextExt
            NamingContextExt ncref = NamingContextExtHelper.narrow(ref);
            String name = "objRemotoPersonal";
            href1 = GestionPersonalIntHelper.narrow(ncref.resolve_str(name));
            // Gestion pacientes
            String name2 = "objRemotoPacientes";
            href2 = GestionPacientesIntHelper.narrow(ncref.resolve_str(name2));

            //Estructura de registro para el callback del Administrador
            POA rootPOA1 = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootPOA1.the_POAManager().activate();
            // instancia el servicio
            AdmCallbackImpl cliente = new AdmCallbackImpl();
            // obtenemos la referencia del servicio & activamos el POAManager
            org.omg.CORBA.Object ref2 = rootPOA1.servant_to_reference(cliente);
            AdmCallbackInt href3 = AdmCallbackIntHelper.narrow(ref2);

            //Estructura de registro para el callback del Fap
            POA rootPOA2 = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootPOA2.the_POAManager().activate();
            // instancia el servicio
            FapCallbackImpl fap = new FapCallbackImpl();
            // obtenemos la referencia del servicio & activamos el POAManager
            org.omg.CORBA.Object ref3 = rootPOA1.servant_to_reference(fap);
            FapCallbackInt href4 = FapCallbackIntHelper.narrow(ref3);

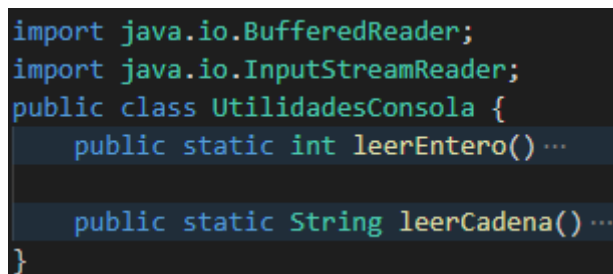
            MenuPrincipal(href1, href2, href3, href4);
        } catch (Exception e) {
            System.out.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}

```

Fig. 31 – Cliente de Objetos

#### d. Implementación Utilidades de consola

Dentro de la carpeta “Utilidades” creamos una nueva clase “UtilidadesConsola.java” e implementamos los métodos para leer un entero y leer una cadena. Imagen de referencia a continuación:



```

import java.io.BufferedReader;
import java.io.InputStreamReader;
public class UtilidadesConsola {
    public static int leerEntero() ...

    public static String leerCadena() ...
}

```

Fig. 32 clase UtilidadesConsola.java

#### e. Implementación Métodos en el Cliente de Objetos

Aquí tendremos definidos luego del *main* los métodos para los menús y la captura de información para posteriormente utilizar las referencias remotas y consumir los servicios de los servidores.

##### 1. Método Menu Principal

Creamos un método estático como se define en la siguiente figura. Este menú será la vista principal al momento de iniciar la aplicación del cliente.



```
private static void MenuPrincipal(GestionPersonalInt ref, GestionPacientesInt ref2,
                                AdminCllbckInt refCllbckAdmin, FapCllbckInt refCllbckFap){
    int opcion = 0;
    do {
        System.out.println("==Menu Principal==");
        System.out.println("1. Abrir Sesión");
        System.out.println("2. Salir");
        System.out.println("=====");
        System.out.print("Ingrese su opción: ");
        opcion = UtilidadesConsole.leerEntero();
        switch (opcion) {
            case 1:
                Login(ref, ref2, refCllbckAdmin, refCllbckFap);
                break;
            case 2:
                System.out.println("Saliendo...");
                break;
            default:
                System.out.println("Opción incorrecta");
        }
    } while (opcion != 2);
}
```

Fig. 33 – MenuPrincipal()

## 2. Método Login

Este método es una fachada para desplegar los menús de los usuarios finales, procesamos las credenciales y con la respuesta del servidor mostramos la vista pertinente dependiendo del tipo de usuario que está haciendo una petición de acceso al servidor. Para el caso del Admin y del PAF se registra el correspondiente callback.

```
private static void login(GestionPersonalInt ref, GestionPacientesInt ref2,
                        AdminCllbckInt refCllbckAdmin, FapCllbckInt refCllbckFap){
    try {
        String user = "";
        String password = "";
        int id = 0;
        boolean login = false;
        System.out.println("==Iniciar Sesión==");
        System.out.println("Usuario: ");
        user = UtilidadesConsole.leerCadena();
        System.out.println("Clave: ");
        password = UtilidadesConsole.leerCadena();
        System.out.println("ID: ");
        id = UtilidadesConsole.leerEntero();
        CredencialID idCredencial = new CredencialID(id, user, password, id);
        login = ref.abrirSesion(idCredencial);
        if (login) {
            boolean resultado;
            personalIDHolder objPersonal = new personalIDHolder();
            //usuarioIDHolder objUsuario = new usuarioIDHolder();
            //res = ref.consultarUsuario(id, objUsuario);
            resultado = ref.consultarPersonal(id, objPersonal);
            if (resultado == true && objPersonal != null) {
                switch (objPersonal.value.occupation) {
                    case "Admin":
                        ref.registrarCallback(refCllbckAdmin);
                        break;
                    case "FAC":
                        ref.registrarCallback(refCllbckFap);
                        break;
                    case "FAC":
                        ref.registrarCallback(refCllbckFap);
                        break;
                }
                Mostrar(varCredencial.id, ref2);
            } else {
                System.out.println("El usuario " + user + " no está autorizado para ingresar al sistema.");
                System.out.println("Verificar con el Usuario y Clave sean correctos");
            }
        } catch (Exception e) {
            System.out.println("Error NO CONTROLADO");
        }
    }
}
```

Fig. 34 – Login()

## 3. Método Menú administrador

Este método despliega el menú para el administrador y en las opciones se delegan a métodos específicos para las opciones disponibles para este usuario.

```

private static void MenuAdmin(GestionPersonalInt ref){
    int opcion = 0;
    do {
        System.out.println("--Menu Admin--");
        System.out.println("1. Registrar Personal ");
        System.out.println("2. Consultar Personal ");
        System.out.println("3. Salir");
        System.out.println("=====");
        System.out.print("Ingrese su opcion: ");
        opcion = UtilidadesConsola.leerEntero();
        switch (opcion) {
            case 1:
                RegistrarPersonal(ref);
                break;
            case 2:
                ConsultarPersonal(ref);
                break;
            case 3:
                System.out.println("Salir...");
                break;
            default:
                System.out.println("Opcion incorrecta");
        }
    } while (opcion != 3);
}

```

Fig. 35 – MenuAdmin()

#### 4. Método Registrar personal

Este método captura por consola los datos de un objeto de tipo personal, luego haciendo uso de la referencia remota consume el servicio respectivo en el servidor de gestión de usuarios. Como este método tiene el control de las restricciones es importante seguir al pie de la letra el siguiente código, que lo pondremos en texto ya que una imagen sería demasiado extensa:

```

private static void RegistrarPersonal(GestionPersonalInt ref){
    try {
        System.out.println("== Registrar Personal ==");
        String tipoID = "";
        int opc = 0;
        do {
            System.out.println("=Tipo de identificacion=");
            System.out.println("1. Cedula de Ciudadania(CC) ");
            System.out.println("2. Tarjeta de Identidad(TI) ");
            System.out.println("3. Pasaporte(PP) ");
            System.out.println("=====");
            System.out.print("Ingrese su opcion: ");

            opc = UtilidadesConsola.leerEntero();

            switch (opc) {
                case 1:
                    tipoID = "CC";
                    break;
                case 2:
                    tipoID = "TI";
                    break;
                case 3:

```



```

        tipoID = "PP";
        break;
    default:
        System.out.println("ADV: Ingrese una opcion
valida");
    }
} while (opc < 1 || opc > 3);
int id = 0;
do{
    System.out.println("Ingrese la Identificacion: ");
    id = UtilidadesConsola.leerEntero();
    if (id < 1) {
        System.out.println("La identificacion debe ser mayot a
cero (0)");
    }
}while(id < 1);
String nombre = "";
do{
    System.out.println("Ingrese El nombre completo: ");
    nombre = UtilidadesConsola.leerCadena();
    if (nombre.length() < 2) {
        System.out.println("El nombre debe contener al menos 2
caracteres");
    }
}while(nombre.length() < 2);
String ocupacion = "";
do {
    System.out.println("== Ocupacion ==");
    System.out.println("1. Secretaria(SEC)  ");
    System.out.println("2. Profesional de Acondicionamiento
Fisico (PAF)  ");
    System.out.println("=====");
    System.out.print("Ingrese su opcion: ");
    opc = UtilidadesConsola.leerEntero();

    switch (opc) {
        case 1:
            ocupacion = "SEC";
            break;
        case 2:
            ocupacion = "PAF";
            break;

        default:
            System.out.println("Seleccione una opcion valida");
    }
}

```

```

    }
    } while (opc < 1 || opc > 2);
    String usuario = "";
    do{
        System.out.println("Digite el usuario: ");
        usuario = UtilidadesConsola.leerCadena();
        if (usuario.length() < 8) {
            System.out.println("El usuario debe contener al menos 8
caracteres!");
        }
    }while(usuario.length() < 2);
    String clave = "";
    do{
        System.out.println("Digite la clave: ");
        clave = UtilidadesConsola.leerCadena();
        if (clave.length() < 8) {
            System.out.println("la clave debe contener al menos 8
caracteres");
        }
    }while(clave.length() < 8);

    personalDTO objPersonal = new personalDTO(tipoID, id, nombre,
ocupacion, usuario, clave);
    BooleanHolder res = new BooleanHolder();
    ref.registrarPersonal(objPersonal, res);
    if (res.value == true) {
        System.out.println("***Personal Registrado
Exitosamente***");
    } else {
        System.out.println("***ERROR: NO se pudo registrar el
Personal***");
    }
} catch (Exception e) {
    System.err.println("ERROR NO CONTROLADO");
}
}

```

### 5. Método Consultar Personal

Este método captura e imprime en consola la respuesta del servidor al consumir el respectivo servicio de consulta de un personal por su ID. Los datos obtenidos son todos los del tipo personal excepto su contraseña:

```

private static void ConsultarPersonal(GestionPersonalInt ref){
    System.out.println("==Consulta de Personal==");
    System.out.println("Ingrese la identificacion: ");
}

```

```

int id = UtilidadesConsola.leerEntero();
try {
    boolean resultado;
    personalDTOHolder objPersonal = new personalDTOHolder();
    resultado = ref.consultarPersonal(id, objPersonal);
    if (resultado) {
        System.out.println();
        System.out.println("==INI Resultado de la consulta==");
        System.out.println("Tipo ID: " + objPersonal.value.tipo_id);
        System.out.println("ID: " + objPersonal.value.id);
        System.out.println("Nombre Completo: " +
objPersonal.value.nombreCompleto);
        System.out.println("Ocupacion: " +
objPersonal.value.ocupacion);
        System.out.println("Usuario: " + objPersonal.value.usuario);
        System.out.println("==FIN Resultado de la consulta==");
        System.out.println();
    }else{
        System.out.println("El personal con id " + id + " no esta
registrado en el sistema!");
    }
} catch (Exception e) {
    System.err.println("El usuario con ID " + id + " no se
encontro!");
}
}

```

#### 6. Método Menú secretaria

Este método despliega el menú para la secretaria y en las opciones se delegan a métodos específicos para las opciones disponibles para este usuario:

```

private static void MenuSec(GestionPersonalInt ref){
    int opcion = 0;
    do {
        System.out.println("==Menu Secretaria==");
        System.out.println("1. Registrar Usuario");
        System.out.println("2. Consultar Usuario");
        System.out.println("3. Salir");
        System.out.println("=====");
        System.out.print("Ingrese su opcion: ");
        opcion = UtilidadesConsola.leerEntero();
        switch (opcion) {
            case 1:
                RegistrarUsuario(ref);
                break;

```

```

        case 2:
            ConsultarUsuario(ref);
            break;
        case 3:
            System.out.println("Salir...");
            break;
        default:
            System.out.println("Opcion incorrecta");
    }
} while (opcion != 3);
}

```

### 7. Método Registrar Usuario

Este método captura por consola todos los datos necesarios para instanciar un objeto de tipo usuario, es importante como en todos los métodos del cliente de objetos seguir los pasos para no cometer errores en la captura y el consumo de los servicios de los servidores

```

private static void RegistrarUsuario(GestionPersonalInt ref){
    try{
        System.out.println("==== Registrar Usuario ====");

        System.out.println("Ingrese el nombre del paciente");
        String nombreUsuario = UtilidadesConsola.leerCadena();
        int id = 0;
        do{
            System.out.println("Ingrese la identificacion");
            id = UtilidadesConsola.leerEntero();
        }while(id < 1);
        String facultad="";
        int opcion = 0;
        do{
            System.out.println("===Facultad a la que pertenece===");
            System.out.println("|1. FACARTES");
            System.out.println("|2. FACAGRO");
            System.out.println("|3. FSALUD");
            System.out.println("|4. FHUMANAS");
            System.out.println("|5. FCCEA");
            System.out.println("|6. FACNED");
            System.out.println("|7. FDERECHO");
            System.out.println("|8. FCIVIL");
            System.out.println("|9. FIET");
            System.out.println("=====");
            System.out.print("Ingrese su opcion: ");

            opcion = UtilidadesConsola.leerEntero();

```

```

    }while(opcion < 1 || opcion > 9);
    switch (opcion) {
        case 1:
            facultad = "FACARTES";
            break;
        case 2:
            facultad = "FACAGRO";
            break;
        case 3:
            facultad = "FSALUD";
            break;
        case 4:
            facultad = "FHUMANAS";
            break;
        case 5:
            facultad = "FCCEA";
            break;
        case 6:
            facultad = "FACNED";
            break;
        case 7:
            facultad = "FDERECHO";
            break;
        case 8:
            facultad = "FCIVIL";
            break;
        case 9:
            facultad = "FIET";
            break;
        default:
            System.out.println("ERROR. La facultad no esta
registrada");
            break;
    }
    do{
        System.out.println("=== Tipo de usuario ===");
        System.out.println("|1. Docente           |");
        System.out.println("|2. Administrativo    |");
        System.out.println("=====");
        System.out.print("Ingrese su opcion: ");
        opcion = UtilidadesConsola.leerEntero();
    }while(opcion < 1 || opcion > 2);
    String tipoUsuario="";
    switch (opcion) {
        case 1:

```

```

        tipoUsuario = "Docente";
        break;
    case 2:
        tipoUsuario = "Administrativo";
        break;
}

Calendar fecha = new GregorianCalendar();
int dia = fecha.get(Calendar.DAY_OF_MONTH);
int mes = fecha.get(Calendar.MONTH) + 1;
int anio = fecha.get(Calendar.YEAR);
String fechaIngreso = dia+"/"+mes+"/"+anio;
do{
    System.out.println("=== Patologia usuario ===");
    System.out.println("|1. Ingresar Patologia  |");
    System.out.println("|2. Ninguna          |");
    System.out.println("=====");
    System.out.print("Ingrese su opcion: ");
    opcion = UtilidadesConsola.leerEntero();
}while(opcion < 1 || opcion > 2);
String patologia="";
switch (opcion) {
    case 1:
        do{
            patologia = UtilidadesConsola.leerCadena();
        }while(patologia.length()<1);
        break;
    case 2:
        patologia = "Ninguna";
        break;
}
String usuario="";
do{
    System.out.println("Ingrese el usuario: ");
    usuario = UtilidadesConsola.leerCadena();
}while(usuario.length() < 8);
String clave="";
do{
    System.out.println("Ingrese la contraseña: ");
    clave = UtilidadesConsola.leerCadena();
}while(clave.length() < 8);

usuarioDTO nuevoUsuario = new usuarioDTO(id, nombreUsuario,
facultad, tipoUsuario, fechaIngreso, patologia, usuario, clave);
BooleanHolder valor = new BooleanHolder();

```

```

        ref.registrarUsuario(nuevoUsuario, valor);//invocacion al metodo
remoto
        if(valor.value)
            System.out.println("Registro realizado
satisfactoriamente...");
        else
            System.out.println("no se pudo realizar el registro...");
    }
    catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    }
}

```

#### 8. Método Consultar Usuario

Este método retorna una consulta del servidor, imprime en consola todos los datos del usuario excepto la contraseña de su nombre de usuario:

```

private static void ConsultarUsuario(GestionPersonalInt ref){
    System.out.println("==== Consultar Usuario ====");
    System.out.println("Ingrese la identificacion: ");
    int id = UtilidadesConsola.leerEntero();

    try {
        boolean resultado;
        usuarioDTOHolder objUsuario = new usuarioDTOHolder();
        resultado = ref.consultarUsuario(id,objUsuario);
        if (resultado) {
            System.out.println("Usuario recuperado exitosamente!");
            System.out.println("=== INFO DEL USUARIO ===!");
            System.out.println("Nombre: " +
objUsuario.value.nombreCompleto);
            System.out.println("ID: " + objUsuario.value.id);
            System.out.println("Tipo: " + objUsuario.value.tipo);
            System.out.println("Facultad: " +
objUsuario.value.facultad);
            System.out.println("Usuario: " + objUsuario.value.usuario);
            System.out.println("Patologia: " +
objUsuario.value.patologia);
            System.out.println("Fecha de Ingreso: " +
objUsuario.value.fechaIngreso);
            System.out.println("==== FIN CONSULTA ====");
        } else {
            System.out.println("El Usuario con id " + id + " no esta
registrado en el sistema");
        }
    }
}

```

```

    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    }
}

```

### 9. Método Menú PAF

Este método despliega el menú para el PAF y en las opciones se delegan a métodos específicos para las opciones disponibles para este usuario:

```

private static void MenuPaf(GestionPersonalInt ref, GestionPacientesInt
ref2){
    int opcion = 0;
    do {
        System.out.println("==Menu PAF==");
        System.out.println("1. Valorar PAF");
        System.out.println("2. Programa Fisico");
        System.out.println("3. Registrar Asistencia");
        System.out.println("4. Salir");
        System.out.println("=====");
        System.out.print("Ingrese su opcion: ");
        opcion = UtilidadesConsola.leerEntero();
        switch (opcion) {
            case 1:
                registrarValoracion(ref, ref2);
                break;
            case 2:
                registrarProgramaFisico(ref, ref2);
                break;
            case 3:
                registrarAsistencia(ref, ref2);
                break;
            case 4:
                System.out.println("Salir...");
                break;
            default:
                System.out.println("Opcion incorrecta");
        }
    } while (opcion != 4);
}

```

### 10. Método Registrar Valoración

Este método captura por consola todos los datos necesarios para instanciar un objeto de tipo valoración, La restricción para valorar a alguien es que el paciente haya sido previamente registrado por la secretaria.



```

private static void registrarValoracion(GestionPersonalInt ref,
GestionPacientesInt ref2){
    try {
        System.out.println("=== Valorar Paciente ===");
        int id=0;
        do{
            System.out.println("Ingrese la identificacion");
            id = UtilidadesConsola.leerEntero();
        }while(id < 1);
        usuarioDTOHolder varObjUsuario = new usuarioDTOHolder();
        if(ref.consultarUsuario(id, varObjUsuario)){
            ValoracionDTOHolder varObjValoracion = new
ValoracionDTOHolder();
            boolean res = ref2.consultarValoracion(id,
varObjValoracion);
            if (!res) {
                Calendar fecha = new GregorianCalendar();
                int dia = fecha.get(Calendar.DAY_OF_MONTH);
                int mes = fecha.get(Calendar.MONTH) + 1;
                int anio = fecha.get(Calendar.YEAR);
                String fechaValoracion = dia+"/"+mes+"/"+anio;
                int fecCarReposo = 0;
                do{
                    System.out.println("Frecuencia Cardiaca en reposo:
");
                    fecCarReposo = UtilidadesConsola.leerEntero();
                }while(fecCarReposo < 1);
                int fecCarActiva = 0;
                do{
                    System.out.println("Frecuencia Cardiaca Activa: ");
                    fecCarActiva = UtilidadesConsola.leerEntero();
                }while(fecCarActiva < 1);
                int estatura = 0;
                do{
                    System.out.println("Estatura(cm): ");
                    estatura = UtilidadesConsola.leerEntero();
                }while(estatura < 1);
                int brazo = 0;
                do{
                    System.out.println("Brazo(cm): ");
                    brazo = UtilidadesConsola.leerEntero();
                }while(brazo < 1);
                int pierna = 0;
                do{
                    System.out.println("Pierna(cm): ");

```

```

        pierna = UtilidadesConsola.leerEntero();
    }while(pierna < 1);
    int pecho = 0;
    do{
        System.out.println("Pecho(cm): ");
        pecho = UtilidadesConsola.leerEntero();
    }while(pecho < 1);
    int cintura = 0;
    do{
        System.out.println("Cintura(cm): ");
        cintura = UtilidadesConsola.leerEntero();
    }while(cintura < 1);
    String estado = "";
    ValoracionDTO objValoracion= new ValoracionDTO(id,
fechaValoracion, fecCarReposo, fecCarActiva, estatura, brazo, pierna, pecho,
cintura, estado);

    BooleanHolder valor = new BooleanHolder();
    ref2.registrarValoracion(objValoracion,
valor);//invocacion al metodo remoto
    if(valor.value)
        System.out.println("Registro realizado
satisfactoriamente...");
    else
        System.out.println("no se pudo realizar el
registro...");
    }else{
        System.out.println("El paciente con id " + id + " ya
tiene una valoracion.");
    }
    }else{
        System.out.println("El paciente con id " + id + " no se
encuentra registrado.");
    }
    } catch (Exception e) {
        System.out.println("ERROR: No controlado");
    }
}
}

```

### 11. Método Registrar Programa

Este método captura por consola todos los datos necesarios para instanciar un objeto de tipo Programa, La restricción para registrar a alguien es que el paciente haya sido previamente valorado por el PAF. Primero capturamos los datos de objetos de tipo Ejercicio y se los asignamos a un objeto de tipo Programa, finalmente una lista de programas conforman el objeto Programa Físico.

```

private static void registrarProgramaFisico(GestionPersonalInt ref,
GestionPacientesInt ref2){
    try {
        System.out.println("==== Programa Fisico ===");
        int id=0;
        do{
            System.out.println("Ingrese la identificacion: ");
            id = UtilidadesConsola.leerEntero();
        }while(id < 1);
        usuarioDTOHolder varObjUsuario = new usuarioDTOHolder();
        if(ref.consultarUsuario(id, varObjUsuario)){
            ProgramaFisicoDTOHolder varObjProgramaFisico = new
ProgramaFisicoDTOHolder();
            if (!ref2.consultarProgramaFisico(id, varObjProgramaFisico))
{
                ValoracionDTOHolder varObjValoracion = new
ValoracionDTOHolder();
                if(ref2.consultarValoracion(id, varObjValoracion)){
                    String fechaIni = "";
                    do {
                        System.out.println("==== Fecha de Inicio del
programa ===");

                        System.out.println("Formato (dd/mm/aa): ");
                        fechaIni = UtilidadesConsola.leerCadena();
                    } while (fechaIni.length() != 8);
                    ProgramaFisicoDTO objProgSemana;
                    ArrayList<ProgramaDTO> objProgramaSemanal = new
ArrayList<ProgramaDTO>();
                    for (int i = 0; i < 3; i++) {
                        String nomEjercicio = "";
                        int repeticiones = 0;
                        int peso = 0;
                        ArrayList<EjercicioDTO> tmp1stEjercicio = new
ArrayList<EjercicioDTO>();
                        ProgramaDTO tmpPrograma;
                        EjercicioDTO tmpEjecercicio;
                        System.out.println("dia " + (i+1) + "de la
semana.");

                        for (int j = 0; j < 3; j++) {
                            do {
                                System.out.println("Nombre del ejercicio
" + (j+1) + ":");

                                nomEjercicio =
UtilidadesConsola.leerCadena();
                            } while (nomEjercicio.length() < 3);

```

```

do {
    System.out.println("Repeticiones: ");
    repeticiones =
UtilidadesConsola.leerEntero();
} while (repeticiones < 1);
do {
    System.out.println("Peso: ");
    peso = UtilidadesConsola.leerEntero();
} while (peso < 3);
tmpEjecercicio = new
EjercicioDTO(nomEjercicio, repeticiones, peso);
tmpLstEjercicio.add(tmpEjecercicio);
}
EjercicioDTO[] varObjEjercicios = new
EjercicioDTO[3];
for (int j = 0; j < tmpLstEjercicio.size(); j++)
{
    varObjEjercicios[j] =
tmpLstEjercicio.get(j);
}
tmpPrograma = new ProgramaDTO((i+1),
varObjEjercicios);
objProgramaSemanal.add(tmpPrograma);
}
ProgramaDTO[] varObjLstPrograma = new
ProgramaDTO[3];
for (int k = 0; k < objProgramaSemanal.size(); k++)
{
    varObjLstPrograma[k] =
objProgramaSemanal.get(k);
}
objProgSemana = new ProgramaFisicoDTO(id, fechaIni,
varObjLstPrograma);

BooleanHolder resultado = new BooleanHolder();
ref2.registrarProgramaFisico(objProgSemana,
resultado);

if (resultado.value) {
    System.out.println("Se registro el programa con
exito");
} else {
    System.out.println("ERROR: No se pudo realizar
el registro");
}
}
}

```

```

        System.out.println("El paciente con id " + id + " no
ha sido valorado.");
    }
    }else{
        System.out.println("El paciente con id " + id + " Ya
tiene un programa registrado");
    }
    }else{
        System.out.println("El paciente con id " + id + " no se
encuentra registrado.");
    }
    } catch (Exception e) {
        System.out.println("ERROR: La oprecion no se pudo completar. " +
e.getMessage());
    }
}
}

```

### 12. Método Registrar Asistencia

Este método captura por consola todos los datos necesarios para instanciar un objeto de tipo Asistencia. La restricción para registrar una asistencia a alguien es que el paciente haya sido previamente valorado por el PAF y se haya registrado su programa físico.

```

private static void registrarAsistencia(GestionPersonalInt ref,
GestionPacientesInt ref2){
    try {
        System.out.println("==== Registrar Asistencia ====");
        int id=0;
        do{
            System.out.println("Ingrese la identificacion: ");
            id = UtilidadesConsola.leerEntero();
        }while(id < 1);
        usuarioDTOHolder varObjUsuario = new usuarioDTOHolder();
        if(ref.consultarUsuario(id, varObjUsuario)){
            ValoracionDTOHolder varObjValoracion = new
ValoracionDTOHolder();
            if(ref2.consultarValoracion(id, varObjValoracion)){
                ProgramaFisicoDTOHolder varObjProgramaFisico = new
ProgramaFisicoDTOHolder();
                if (ref2.consultarProgramaFisico(id,
varObjProgramaFisico)) {
                    String fechaAsistencia = "";
                    do {
                        System.out.println("==== Fecha de asistencia
====");
                        System.out.println("Formato (dd/mm/aa): ");

```

```

        fechaAsistencia =
UtilidadesConsola.leerCadena();
    } while (fechaAsistencia.length() != 8);
    String observacion = "";
    int opcion=0;
    do
    {
        System.out.println("=== Observacion ===");
        System.out.println("|1. Ingresar
excusa");

        System.out.println("|2. No asiste");
        System.out.println("|3. Ninguna");
        System.out.println("=====
");

        System.out.print("Ingrese su opcion: ");
        opcion = UtilidadesConsola.leerEntero();
    }while(opcion < 1 || opcion > 3);
    switch (opcion) {
        case 1:
            System.out.print("Ingrese la excusa: ");
            observacion =
UtilidadesConsola.leerCadena();
            break;
        case 2:
            observacion = "No asiste";
            break;
        case 3:
            observacion = "Ninguna";
            break;
    }
    AsistenciaDTO objAsistencia = new AsistenciaDTO(id,
fechaAsistencia, observacion);
    BooleanHolder resultado = new BooleanHolder();
    ref2.registrarAsistencia(objAsistencia,resultado);
    IntHolder contador = new IntHolder();
    ref2.contarFaltas(id, contador);
    if (resultado.value) {
        System.out.println("Se registro la asistencia
con exito!");
        if (contador.value >= 3) {
            System.out.println("Se han exedido el numero
de faltas.");
            System.out.println("El usuario con id " + id
+ " tendra que volver a rezalizar valoracion,\ny elaborar el programa
fisico.");

```

```

        }
    } else {
        System.out.println("ERROR :No se pudo registrar
la asistencia.");
    }
} else {
    System.out.println("El paciente con id " + id + "
aun no tiene registrado el programa fisico!");
}
} else {
    System.out.println("El paciente con id " + id + " aun no
ha sido valorado!");
}
}
} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());
}
}
}

```

### 13. Método Menú Usuario

Este método despliega el menú para el usuario-paciente y en las opciones se delegan a métodos específicos para las opciones disponibles para este usuario:

```

private static void MenuUser(int id, GestionPacientesInt ref){
    int opcion = 0;
    do{
        System.out.println("==== Menu Usuario =====");
        System.out.println("|1. Consultar Valoracion");
        System.out.println("|2. Consultar
Programa");
        System.out.println("|3. Consultar asistencia");
        System.out.println("|4. Salir");
        System.out.println("=====");
        System.out.println("Ingrese su opcion: ");
        opcion = UtilidadesConsola.leerEntero();
        switch(opcion)
        {
            case 1:
                ConsultarValoracion(id, ref);
                break;
            case 2:
                ConsultarPrograma(id, ref);
                break;
            case 3:
                ConsultarAsistencia(id, ref);

```

```

        break;
    case 4:
        System.out.println("Saliendo...");
        break;
    default:
        System.out.println("Opcion incorrecta");
    }
}while(opcion != 4);
}

```

#### 14. Método Consultar Valoración

Este método captura e imprime en consola la respuesta del servidor al consumir el respectivo servicio de consulta de valoración para el usuario actual que ha iniciado sesión.

```

private static void ConsultarValoracion(int id, GestionPacientesInt ref){
    try {
        System.out.println("==== Consultar Valoracion ====");
        System.out.println("==== Valoracion del paciente " + id + " ====");
        System.out.println("==== Resultados de consulta ====");
        ValoracionDTOHolder varObjValoracion = new ValoracionDTOHolder();
        boolean resultado = ref.consultarValoracion(id, varObjValoracion);
        if (resultado) {
            System.out.println("Fecha de valoracion: " + varObjValoracion.value.fechaValoracion);
            System.out.println("Frecuencia cardiaca en reposo: " + varObjValoracion.value.fecCardiacaReposo);
            System.out.println("Frecuencia cardiaca activa: " + varObjValoracion.value.fecCardiacaActiva);
            System.out.println("Estatura: " + varObjValoracion.value.estatura + " cm.");
            System.out.println("Medida del brazo: " + varObjValoracion.value.brazo + " cm.");
            System.out.println("Medida de la pierna: " + varObjValoracion.value.pierna + " cm.");
            System.out.println("Medida del pecho: " + varObjValoracion.value.pecho + " cm.");
            System.out.println("Medida de la cintura: " + varObjValoracion.value.cintura + " cm.");
            System.out.println("Estado: " + varObjValoracion.value.estado);
        }else{
            System.out.println("No hay datos para su consulta!");
        }
    } catch (Exception e) {
        System.out.println("ERROR: " + e.getMessage());
    }
}

```

#### 15. Método Consultar Programa Físico

Este método captura e imprime en consola la respuesta del servidor al consumir el respectivo servicio de consulta de Programa Físico para el usuario actual que ha iniciado sesión.

```

private static void ConsultarPrograma(int id, GestionPacientesInt ref){
    try {
        System.out.println("==== Consultar Programa Fisico ====");
        System.out.println("==== Programa del paciente " + id + "
====");
        System.out.println("==== Resultados de consulta ====");
        ProgramaFisicoDTOHolder varObjProgramaFisico = new
ProgramaFisicoDTOHolder();
        boolean resultado = ref.consultarProgramaFisico(id,
varObjProgramaFisico);
    }
}

```



```

        if (resultado) {
            System.out.println("Fecha de inicio: " +
varObjProgramaFisico.value.fechaInicio);
            System.out.println("====Programa de la semana====");
            for (int i = 0; i <
varObjProgramaFisico.value.listaProgramaSemana.length; i++) {
                System.out.println("-----");
                System.out.println("dia de la semana: " +
varObjProgramaFisico.value.listaProgramaSemana[i].dia);
                for (int j = 0; j <
varObjProgramaFisico.value.listaProgramaSemana[i].listaEjercicios.length;
j++) {
                    System.out.println("Nombre del ejercicio: " +
varObjProgramaFisico.value.listaProgramaSemana[i].listaEjercicios[j].nombreE
jercicio);
                    System.out.println("Repeticiones a realizar: " +
varObjProgramaFisico.value.listaProgramaSemana[i].listaEjercicios[j].repetic
iones);
                    System.out.println("Peso: " +
varObjProgramaFisico.value.listaProgramaSemana[i].listaEjercicios[j].peso +
" Kg");
                    System.out.println("-----");
                }
            }
        }else{
            System.out.println("No hay datos para su consulta!");
        }
        System.out.println("=====");
    } catch (Exception e) {
        System.out.println("ERROR:" + e.getMessage());
    }
}
}

```

### 16. Método Consultar Programa Físico

Este método captura e imprime en consola la respuesta del servidor al consumir el respectivo servicio de consulta de Asistencia para el usuario actual que ha iniciado sesión.

```

private static void ConsultarAsistencia(int id, GestionPacientesInt ref){
    try {
        System.out.println("==== Consultar Asistencia ====");
        System.out.println("=== Asistencia del paciente con id " + id +
" ===");
        System.out.println("==== Resultados de consulta ====");
        ValoracionDTOHolder varObjValoracion = new
ValoracionDTOHolder();
    }
}

```

```

        if (ref.consultarValoracion(id, varObjValoracion)) {
            ProgramaFisicoDTOHolder varObjProgramaFisico = new
ProgramaFisicoDTOHolder();
            if (ref.consultarProgramaFisico(id, varObjProgramaFisico)) {
                ArrayAsistenciaHolder lstAsistencia = new
ArrayAsistenciaHolder();
                boolean resultado = ref.consultarAsistencia(id,
lstAsistencia);
                if (resultado) {
                    for (int i = 0; i < lstAsistencia.value.length; i++)
{
                        System.out.println("Registro numero: " + (i+1));
                        System.out.println("Fecha de Asistencia: " +
lstAsistencia.value[i].fechaAsistencia);
                        System.out.println("Observacion: " +
lstAsistencia.value[i].observacion);
                        System.out.println("=====
=====");
                        System.out.println("");
                    }
                }else{
                    System.out.println("No posee registro de
asistencia!");
                }
            }else{
                System.out.println("Para acceder al registro el PAF debe
primero elaborar su programa!");
            }
        }else{
            System.out.println("Para acceder al registro debe ser
valorado primero!");
        }
    } catch (Exception e) {
        System.out.println("ERROR: " + e.getMessage());
    }
}

```

## V. Descripción de compilación y ejecución

Para compilar nuestro proyecto nos ubicamos en la carpeta “src”, y a continuación compilamos cada uno de los servidores y el cliente de objetos con los comandos siguientes:

- Compilar cliente
  - `javac -d ../bin cliente/*.java`

- Compilar s\_seguimiento\_usuarios
  - `javac -d ../bin s_seguimiento_usuarios/sop_corba/*.java`
  - `javac -d ../bin s_seguimiento_usuarios/servidor/*.java`
- Compilar s\_gestion\_usuarios
  - `javac -d ../bin s_gestion_usuarios/sop_corba/*.java`
  - `javac -d ../bin s_gestion_usuarios/servidor/*.java`

Para ejecutar la aplicación se deben seguir los siguientes pasos, primero se lanza el servidor de objetos 2, luego el servidor de objetos 1 y por último el cliente. Las operaciones descritas anteriormente se realizan con los siguientes comandos:

Esta vez nos ubicamos en la carpeta “bin” de nuestro proyecto.

- Lanzar el ORBD
  - `orbd -ORBInitialHost localhost -ORBInitialPort 2020`
- Ejecutar Servidor de objetos 2
  - `java s_seguimiento_usuarios.servidor.ServidorDeObjetos -ORBInitialHost localhost -ORBInitialPort 2020`
- Ejecutar Servidor de objetos 1
  - `java s_gestion_usuarios.servidor.ServidorDeObjetos -ORBInitialHost localhost -ORBInitialPort 2020`
- Ejecutar Cliente de objetos
  - `java cliente.ClienteDeObjetos -ORBInitialHost localhost -ORBInitialPort 2020`

Cada ejecución requiere que ingresemos una dirección IP, en este momento ingresamos “localhost”, también nos solicitará un número de puerto, aquí ingresamos un número superior a 1024, en todos los casos debemos ingresar los mismos datos para tener una conexión exitosa.

## VI. Repositorio GitHub

Repositorio de GitHub: [AppDistribuida-Corba-Java](#)